18. Hinweise zur schriftlichen Abiturprüfung 2021 im Fach Informatik

A. Fachbezogene Hinweise

Grundlage für die schriftliche Abiturprüfung 2021 in Niedersachsen sind die Einheitlichen Prüfungsanforderungen in der Abiturprüfung Informatik (EPA vom 01.12.1989 i.d.F. vom 05.02.2004), konkretisiert durch das Kerncurriculum Informatik für das Gymnasium – gymnasiale Oberstufe, die Gesamtschule – gymnasiale Oberstufe und das Kolleg (KC, 2017).

Entsprechend der Ausführungen des Kerncurriculums zu einheitlichen Darstellungsformen und Funktionsumfängen (KC, 2017, Abschnitt 3.5) sind die durch die „Ergänzenden Hinweise zum Kerncurriculum Informatik für die gymnasiale Oberstufe am Gymnasium, an der Gesamtschule sowie für das Kolleg“ mit Stand vom Juni 2018 vorgenommenen Festlegungen für die schriftliche Abiturprüfung 2021 verbindlich zu berücksichtigen.

B. Hinweise zu den Prüfungsaufgaben

Sämtliche im Kerncurriculum (KC, 2017) genannten inhaltlichen und prozessorientierten Kompetenzen sind für die schriftliche Abiturprüfung verbindlich. Dies gilt insbesondere auch für die Kompetenzen, die in den Lernfeldern für die Einführungsphase ausgewiesen sind.

Jede **Prüfungsaufgabe** besteht aus drei Aufgaben. Den Prüflingen werden jeweils Aufgabenvorschläge aus zwei Themenblöcken (1 und 2) zur Auswahl vorgelegt. In jeder Prüfungsaufgabe werden alle drei Lernfelder abgebildet. Aus jedem Block ist genau ein Aufgabenvorschlag zur Bearbeitung auszuwählen. Andere Kombinationen sind nicht zulässig.
Jeder Vorschlag aus Block 1 enthält eine Aufgabe, die 50% der insgesamt zu erreichenden Bewertungseinheiten (BE) umfasst. Jeder Vorschlag aus Block 2 enthält zwei Aufgaben, die zusammen ebenfalls 50% der insgesamt zu erreichenden BE umfassen.

Block 1 (50% der BE)	**Block 2 (50% der BE)**
Vorschlag 1A mit einer Aufgabe	Vorschlag 2A mit 2 Aufgaben
Vorschlag 1B mit einer Aufgabe	Vorschlag 2B mit 2 Aufgaben

C. Sonstige Hinweise

- Die Aufgabenstellungen enthalten keinen Code in einer konkreten Programmiersprache.
- Aufgaben, die die Implementierung in einer konkreten Programmiersprache erfordern, sind von den Schülerinnen und Schülern in Java oder einer anderen objektorientierten Sprache mit imperativem Kern zu bearbeiten.
- Es werden keine Aufgaben gestellt, für die der Einsatz eines Rechners erforderlich ist.

Hilfsmittel

- Die in Anlage 1 dokumentierten Ausführungen zu Operationen für Zeichenketten, Operationen für dynamische Reihungen, Stapel, Schlangen und Binärbäume sowie zum Sprachumfang von Datenbankabfragen sind in ausgedruckter Form als Hilfsmittel im Fach Informatik zulässig.
- Die Verwendung eines Taschenrechners oder einer Formelsammlung ist in der schriftlichen Abiturprüfung im Fach Informatik nicht zulässig

Anlage 1

Hilfsmittel für Prüflinge zur Verwendung in der schriftlichen Abiturprüfung im Fach Informatik

1 Operationen für Zeichenketten

- Bestimmen der Länge einer Zeichenkette
- Auslesen eines Zeichens an einer bestimmten Position
- Ersetzen eines Zeichens an einer bestimmten Position
- Verbinden von zwei Zeichenketten zu einer
- Prüfen des Inhalts von zwei Zeichenketten auf Gleichheit
- Lexikographisches Vergleichen von zwei Zeichenketten

2 Operationen für dynamische Reihungen, Stapel, Schlangen und Binärbäume

Die Klassen verwenden Inhaltstypen (bzw. Inhaltsklassen), die jeweils der aktuellen Aufgabenstellung angepasst werden. Mögliche Laufzeitfehler bei der Anwendung der Operationen, z. B. Entnehmen bei einem leeren Stapel oder Zugriff auf eine nicht existierende Position einer dynamischen Reihung, müssen bei der Verwendung durch entsprechende Abfragen explizit ausgeschlossen werden.

Dynamische Reihung

Die Nummerierung der Elemente der dynamischen Reihung beginnt mit dem Index 0.

`DynArray()`
Eine leere dynamische Reihung wird angelegt.

`isEmpty(): Wahrheitswert`
Wenn die dynamische Reihung kein Element enthält, wird der Wert *wahr* zurückgegeben, sonst der Wert *falsch*.

`getItem(index: Ganzzahl): Inhaltstyp`
Der Inhalt des Elements an der Position `index` wird zurückgegeben.

`append(inhalt: Inhaltstyp)`
Ein neues Element mit dem übergebenen Inhalt wird am Ende der dynamischen Reihung angefügt.

`insertAt(index: Ganzzahl, inhalt: Inhaltstyp)`
Ein neues Element mit dem übergebenen Inhalt wird an der Position `index` in die dynamische Reihung eingefügt. Das Element, das sich vorher an dieser Position befunden hat, und alle nachfolgenden werden nach hinten verschoben. Entspricht der Wert von `index` der Länge der dynamischen Reihung, so wird ein neues Element am Ende der dynamischen Reihung angefügt.

`setItem(index: Ganzzahl, inhalt: Inhaltstyp)`
Der Inhalt des Elementes an der Position `index` wird durch den übergebenen Inhalt ersetzt.

`delete(index: Ganzzahl)`
Das Element an der Position `index` wird entfernt. Alle folgenden Elemente werden um eine Position nach vorne geschoben.

`getLength(): Ganzzahl`
Die Anzahl der Elemente der dynamischen Reihung wird zurückgegeben.

Stapel

`Stack()`
Ein leerer Stapel wird angelegt.

`isEmpty(): Wahrheitswert`
Wenn der Stapel kein Element enthält, wird der Wert *wahr* zurückgegeben, sonst der Wert *falsch*.

`top(): Inhaltstyp`
Der Inhalt des obersten Elements des Stapels wird zurückgegeben, das Element aber nicht entfernt.

`push(inhalt: Inhaltstyp)`
Ein neues Element mit dem übergebenen Inhalt wird auf den Stapel gelegt.

`pop(): Inhaltstyp`
Der Inhalt des obersten Elements wird zurückgegeben und das Element wird aus dem Stapel entfernt.

Schlange

`Queue()`
Eine leere Schlange wird angelegt.

`isEmpty(): Wahrheitswert`
Wenn die Schlange kein Element enthält, wird der Wert *wahr* zurückgegeben, sonst der Wert *falsch*.

`head(): Inhaltstyp`
Der Inhalt des ersten Elements der Schlange wird zurückgegeben, das Element aber nicht entfernt.

`enqueue(inhalt: Inhaltstyp)`
Ein neues Element mit dem angegebenen Inhalt wird am Ende an die Schlange angehängt.

`dequeue(): Inhaltstyp`
Der Inhalt des ersten Elements wird zurückgegeben und das Element wird aus der Schlange entfernt.

Binärbaum

`BinTree()`
Ein Baum wird erzeugt. Der Baum besitzt keine Teilbäume. Die Wurzel besitzt keinen Inhaltswert.

`BinTree(inhalt: Inhaltstyp)`
Ein Baum wird erzeugt. Der Baum besitzt keine Teilbäume. Die Wurzel erhält den übergebenen Inhalt als Wert.

`hasItem(): Wahrheitswert`
Wenn die Wurzel des Baums einen Inhaltswert besitzt, wird der Wert *wahr* zurückgegeben, sonst der Wert *falsch*.

`getItem(): Inhaltstyp`
Die Operation gibt den Inhaltswert der Wurzel des Baums zurück.

`setItem(inhalt: Inhaltstyp)`
Die Wurzel des Baums erhält den übergebenen Inhalt als Wert.

`deleteItem()`
Die Operation löscht den Inhaltswert der Wurzel des Baums.

`isLeaf(): Wahrheitswert`
Wenn der Baum keine Teilbäume besitzt, die Wurzel des Baums also ein Blatt ist, wird der Wert *wahr* zurückgegeben, sonst der Wert *falsch*.

`hasLeft(): Wahrheitswert`
Wenn der Baum einen linken Teilbaum besitzt, wird der Wert *wahr* zurückgegeben, sonst der Wert *falsch*.

`getLeft(): Binärbaum`
Die Operation gibt den linken Teilbaum zurück.

`setLeft(b: Binärbaum)`
Der übergebene Baum wird als linker Teilbaum gesetzt.

`deleteLeft()`
Die Operation löscht den linken Teilbaum.

`hasRight(): Wahrheitswert`
Wenn der Baum einen rechten Teilbaum besitzt, wird der Wert *wahr* zurückgegeben, sonst der Wert *falsch*.

`getRight(): Binärbaum`
Die Operation gibt den rechten Teilbaum zurück.

`setRight(b: Binärbaum)`
Der übergebene Baum wird als rechter Teilbaum gesetzt.

`deleteRight()`
Die Operation löscht den rechten Teilbaum.

3 Sprachumfang von Datenbankabfragen

SELECT-Anweisung:

SELECT [DISTINCT | ALL] * | $spalte_1$ [AS $alias_1$], $spalte_2$ [AS $alias_2$], …, $spalte_n$ [AS $alias_n$]

FROM $tabelle_1$, $tabelle_2$, …, $tabelle_m$

[**WHERE** $bedingung_1$ (AND | OR) $bedingung_2$ … (AND|OR) $bedingung_k$]

[**GROUP BY** $spalte_1$, $spalte_2$, … , $spalte_1$]

[**HAVING** $gruppenBedingung_1$ (AND | OR) $gruppenBedingung_2$ … (AND | OR) $gruppenBedingung_s$]

[**ORDER BY** $spalte_1$ [ASC | DESC], $spalte_2$ [ASC | DESC], ..., $spalte_t$ [ASC | DESC]]

[**LIMIT** anzahl]

Angaben in eckigen Klammern sind optional. Spalten können Attribute, Aggregatfunktionen oder Berechnungen sein. Bei `GROUP BY` und `ORDER BY` ist auch die Angabe eines Alias möglich.

Operatoren für Berechnungen: +, –, *, /

Operatoren für Vergleiche in Bedingungen: =, != (ungleich), >, <, >=, <=, `NOT`, `LIKE` (mit den Platzhaltern _ und `%`) , `BETWEEN`, `IN`, `IS NULL`

Aggregatfunktionen: `AVG( )`, `COUNT( )`, `MAX( )`, `MIN( )`, `SUM( )`